The Handling of Graphs on PC Clusters: A Coarse Grained Approach

Isabelle Guérin Lassous, Jens Gustedt, Michel Morvan

▶ **To cite this version:**

Isabelle Guérin Lassous, Jens Gustedt, Michel Morvan. The Handling of Graphs on PC Clusters: A Coarse Grained Approach. [Research Report] RR-3897, INRIA. 2000. inria-00072757

HAL Id: inria-00072757
https://inria.hal.science/inria-00072757

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The Handling of Graphs on PC Clusters: A Coarse Grained Approach

Isabelle Guérin Lassous — Jens Gustedt — Michel Morvan

N° 3897

Mars 2000

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--3897--FR+ENG

The Handling of Graphs on PC Clusters: A Coarse Grained Approach

Isabelle Guérin Lassous* , Jens Gustedt† , Michel Morvan‡

Thème 1 —Réseaux et systèmes
Projet Hipercom

Rapport de recherche n° 3897 —Mars 2000 —12 pages

Abstract: We study the relationship between the design and analysis of graph algorithms in the coarsed grained parallel models and the behavior of the resulting code on clusters. We conclude that the coarse grained multicomputer model (CGM) is well suited to design competitive algorithms, and that it is thereby now possible to aim to develop portable, predictable and efficient parallel code for graph problems on clusters.

Key-words: parallel graph algorithms, parallel models, CGM, clusters, experiments

* INRIA Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay, France. Phone: +33 1 39 63 59 08. Email: Isabelle.Guerin-Lassous@inria.fr.

† LORIA and INRIA Lorraine, Campus scientifique, BP 239, 54506 Vandœuvre-lès-Nancy, France. Phone: +33 3 83 59 30 90. Email: Jens.Gustedt@loria.fr.

‡ LIAFA and Université Paris 7 and Institut Universitaire de France, Case 7014, 2, place Jussieu, 75251 Paris Cedex 05, France. Phone: +33 1 44 27 28 42. Email: morvan@iafa.jussieu.fr.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 —Télécopie : +33 1 39 63 53 30

Manipulation des graphes sur des grappes de PCs : une approche à gros grain

Résumé : Nous étudions la relation entre la conception et l'analyse d'algorithmes sur les graphes dans les modèles parallèles à gros grain et le comportement du code résultant sur grappes. Nous concluons que le modèle à gros grain CGM est bien adapté pour concevoir des algorithmes concurrentiels, et qu'il est maintenant possible de développer du code parallèle portable, prédictif et efficace sur des grappes pour les problèmes liés aux graphes.

Mots-clés : algorithmes parallèles de graphes, modèles parallèles, CGM , grappes, analyse expérimentale

1 Introduction and Overview

Graphs are basic tools in computer science and many problems in this field resort to graph algorithms. Some of these algorithms have a very high complexity or handle too many objects such that the problems can not be solved with only one single computer. Parallel/distributed computation is intended to attack these different problems simultaneously.

For this reason, parallel graph algorithms are a field that had a rich development since the early beginning of parallel computation. But if there are a lot of theoretical studies in this area, relatively few implementations have been presented for all these algorithms that were designed. Moreover most of these implementations have been carried out on specific parallel machines (C90, T3E, CM2, CM5, MasPar, Paragon, ...) using special purpose hardware.

Recently, the availability of high-speed networks, high-performance microprocessors, off-the-shelf hardware and portable software components are making networks of computers effective parallel/distributed systems. Due to their low cost and complexity, such clusters seem to lead to a much wider acceptance of parallel/distibuted computing.

In this paper, we address ourselves to the problem of graph handling on clusters. Two questions were the starting point of our work: Which parallel model allows to develop algorithms that are:

feasible: they can be implemented with a reasonable effort,

portable: the code can be used on different platforms without rewriting it,

predictable: the theoretical analysis allows the prediction of the behavior in real platforms

efficient: the code runs correctly and is more efficient than the sequential code?

What are the possibilities and limits of graphs handling in real parallel platforms? This work comes within a wider framework concerning parallel/distributed machines, but in this paper we focus on clusters. Our approach is original in the sense that it consists in proposing new algorithms and in studying the behavior of these algorithms as well as other ones from the literature by experiments on clusters.

Due to space limitations, we omit algorithm descriptions and only present the results and analysis of some experiments on PC clusters. For each tackled problem, we will briefly describe the algorithm and give pointers for more details.

Section 2 presents the chosen model CGM (Coarse Grained Multicomputer) that seems well adapted for computations on clusters. Section 3 presents the experimental framework. Section 4 gives the results obtained for one of the basic problem that is sorting. Section 5 deals with the difficult problem of list ranking. Section 6 shows that it is possible to solve the connected components problem on dense graph efficiently. Section 7 shows that an algorithm with $\log p$ supersteps (p is the number of processors) can be efficient in practise. Section 8 will conclude on the use of the coarse grained models and especially of CGM for clusters.

2 The parallel model

The first parallel models that have been proposed are fine grained models. In these models, it is supposed that $p \in \theta(n)$ where n is the size of the problem and p the number of processors. Among the fine grained models, two main classes have emerged: the

PRAM model (Parallel Random Access Machine), see [Karp and Ramachandran, 1990], and the distributed memory machines models, see [Leighton, 1992]. If the PRAM model is very useful to point out the possible parallelism of a problem, it is quite unrealistic. The distributed memory models are more realistic, but are too close to the network structure of the considered machine to lead to portable code.

Clusters as most of the current parallel machines are coarse grained, that is every processor has a substantial local memory. Recently, several works tried to provide models that take realistic characteristics of existing platforms into account while covering at the same time as many parallel platforms as possible.

Proposed by [Valiant, 1990], BSP (Bulk Synchronous Parallel) is the originating source of this family of models. It formalizes the architectural features of existing platforms in very few parameters. The LogP model proposed by [Culler et al., 1993] considers more architectural details compared to BSP, whereas the CGM model (Coarse Grained Multicomputer) initiated by [Dehne et al., 1993] is a simplification of BSP. We chose CGM because it has a high abstraction that easily enables the design of algorithms and offered the simplest realization of the goals we had in mind. One aim of our work is to justify whether this model is well suited to handle graphs on clusters.

The three models of parallel computation have a common machine model: a set of processors that is interconnected by a network. A processor can be a monoprocessor machine, a processor of a multiprocessors machine or a multiprocessors machine. The network can be any communication medium between the processors (bus, shared memory, Ethernet, etc).

CGM The CGM model describes the number of data per processor explicitly. Indeed, for a problem of size n, it assumes that the processors can hold $O(\frac{n}{p})$ data in their local memory and that $1 \ll \frac{n}{p}$. Usually the later requirement is put in concrete terms by assuming that $p \leq \frac{n}{p}$ because each processor has to store information about the other processors.

The algorithms are an alternation of supersteps. In a superstep, a processor can send or receive once to and from each other processor and the amount of data exchanged in a superstep by one processor in total is at most $O(\frac{n}{p})$. Unlike BSP, the supersteps are not assumed to be synchronized explicitly. Such a synchronization is done implicitly during the communications steps.

In CGM we have to ensure that the number R of supersteps is particularly small compared to the size of the input. For instance, we can ensure that R is a function that only depends on p (and not on n the size of the input).

3 Implementation background

We have implemented these algorithms on two PC clusters. Note that our code also ran on distributed/shared memory parallel machines. The use of two clusters with different interconnection networks should allow to check that the code does not depend too much on the underlying network.

The first cluster[1] consists of 13 *PentiumPro* 200 PCs with 128 MB memory each. The PCs are interconnected by a 100 Mb/s full-duplex *Fast Ethernet* network, see [Tanenbaum, 1997], having 93 μs latency. In the following, we refer to it as *PF*. The sec-

[1] http://www.inria.fr/sophia/parallel

ond cluster[2] consists of 12 *PentiumPro* 200 PCs with 64 MB of memory each. The interconnection network is a *Myrinet*[3] network of 1.28 Gb/s and with 5 μs latency. We refer to it as *POPC*. The programming language is C++ (gcc) and the communication libraries are PVM and MPI.

An outline of the analysis All the tests have been carried out ten times for each input size. The results given are an average of ten tests. All the execution times are in seconds. Each execution time is taken as the maximum value of the execution times obtained on each of the p processors.

In all the given figures, the x-axis corresponds to n, the input size and the y-axis gives the execution time in *seconds per elements*. Both scales are logarithmic.

To test and instrument our code we generated input objects randomly. Most of these objects were constructed from random permutations. See [Guérin Lassous, 1999, Guérin Lassous and Thierry, 2000] for more details. The time required for the generation of an object is not included in the times as they are presented.

4 A basic operation: sorting

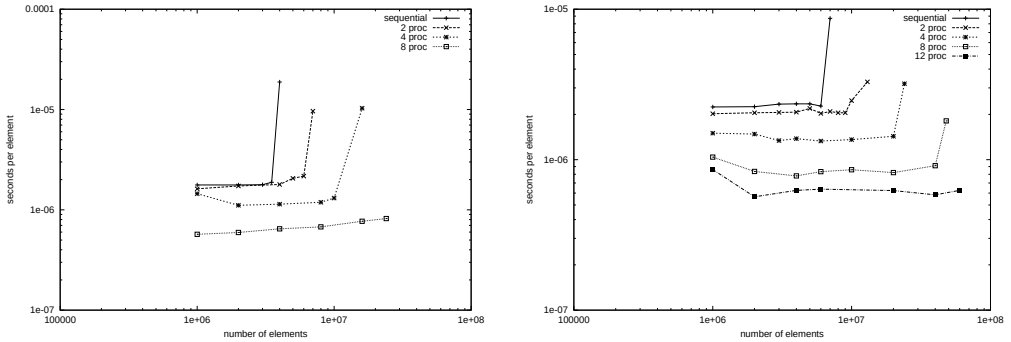
The choice of the sorting algorithm is a critical point due to its widespread use to solve graph problems. In the BSP model, there are deterministic as well as randomized algorithms. In the CGM setting, all these algorithms translate to have a constant number of supersteps. The algorithm proposed by [Goodrich, 1996] is theoretically the most performing, but is complicated to implement and quite greedy in its use of memory.

We chose the algorithm of [Gerbessiotis and Valiant, 1994] because it is conceptually simple and requires only 3 supersteps. It is based on the sample technique which uses $p-1$ splitters to cut the input elements in p packets. The choice of the splitters is then essential to ensure that the packets have more or less the same size. The algorithm is randomized and bounds the packets size by $\left(1+\frac{1}{\sqrt{\ln n}}\right)\left(\frac{n-p+1}{p}\right)$ with high probability, only. For more details on this algorithm.

In our implementation, the sorted integers are the standard 32 bit `int` types of the machines. We use counting sort, [Cormen et al., 1990], as the sequential sorting subroutine. To distribute the data according to the splitters, we do a dichotomic search on $p-1$ to find the destination packet of each element. By that we only introduce a $\log p$ factor. Therefore, this sort can be solved with probability $1-o(1)$ in $O(T_S(\frac{n}{p})+\frac{n}{p}\lceil\log(p-1)\rceil)$ local computations on each processor and with 3 supersteps where T_S is the complexity of the sequential sort.

Figure 1 gives the execution times per element of the program with $1,2,4$ and 8 PC for POPC, and with $1,2,4,8$ and 12 PC for PF. The right ends of the curves for the execution times demonstrate the swapping effects. Measures begin at one million elements to satisfy some inequalities given by this sort. The memory of an individual PC in PF is two times larger than the one for POPC, therefore PF can sort two times more data. As expected, we see that the curves (besides swapping) are near constant in n and the execution times are neatly improved when we use $2,4,8$ or 12 PC. We see that this parallel sort can handle very large data efficiently, whereas the sequential algorithm is

[2] http://www.ens-lyon.fr/LHPC/ANGLAIS/popc.html
[3] http://www.myri.com/



(a) POPC (b) PF

Figure 1: Sorting

stuck quite early due to the swapping effects. Note that PF can sort 76 million integers with 12 PC in less than 40 seconds.

5 The list ranking problem

The list ranking problem frequently occurs in parallel algorithms that use dynamic objects like lists, trees or graphs. The problem is the following: given a linked list of elements, for each element x we want to know the distance from x to the tail of the list. If it is easy to solve it sequentially, it seems much more difficult in parallel.

The first proposed algorithms were formulated in the PRAM model. In the coarse grained models, several algorithms were also proposed, but none of them is optimal, see [Dehne and Song, 1996] and [Caceres et al., 1997]. As far as we know, few implementations have been realized, and none of them runs on clusters and seems to be portable, see [Reid-Miller, 1994, Sibeyn et al., 1999].

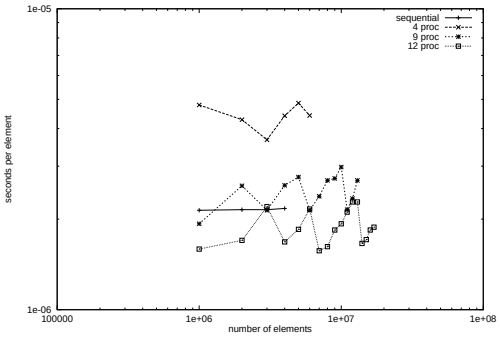


Figure 2: List Ranking on POPC

[Guérin Lassous and Gustedt, 2000] proposed a randomized algorithm that uses the technique of *independent sets*, as described in [Jájá, 1992]. It requires $O(\log p)$ supersteps and $O(n)$ for the total communication cost and local computations.

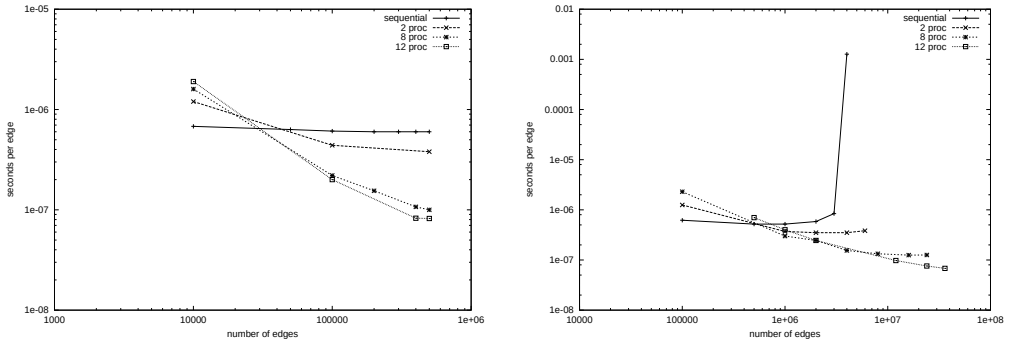
To not overload the study of the results, we only present the experiments on POPC, but the results on PF are alike. Figure 5 gives the execution times *per element* in function of the list

size. p varies from 4 to 12, because the memory of the processors is saturated when we use 2 or 3 PC. All the curves stop before the memory saturation of the processors. We start the measures for lists with 1 million elements, because for smaller size, the sequential algorithm performs so well that using more processors is not very useful. A positive fact that we can deduce from the plots given in Figure 5 is that the execution time for a fixed amount of processors p shows a linear behavior as expected. One might get the impression from Figure 5 that it deviates a bit from linearity in n, but this is only a scaling effect: the variation between the values for a fixed p and n varying is very small (less than $1\mu s$). We see that from 9 PC the parallel algorithm becomes faster than the sequential one. The parallel execution time decreases also with the number of used PC. Nevertheless, the speedups (the ratio between the sequential time and the parallel time) are quite restricted. We see also that this algorithm performs well on huge lists. Due to the swapping effects, the sequential algorithm dramatically changes its behavior when run with more than 4 million elements. For 5 millions elements, the execution time is a little bit higher than 3000 seconds, whereas 12 PC solve the problem in 9.24 seconds. We see also that we only need 18 seconds to handle lists with 17 millions elements.

6 Connected Components for dense graphs

Searching for the connected components of a graph is also a basic graph operation. For a review of the different PRAM algorithms on the subject see [Jájá, 1992]. Few algorithms for the coarse grained models have been proposed. In [Caceres et al., 1997], the first deterministic CGM algorithm is presented. It requires $O(\log p)$ supersteps and is based on PRAM simulations and list ranking. According to our experience, it seems that the simulation of PRAM algorithms is complex to implement, computationally complex in practice and predictable with difficulty. Moreover, this algorithm uses the list ranking that is really a challenging problem as shown previously. On the other hand, the part of the algorithm of [Caceres et al., 1997] that is specific to CGM doesn't have these constraints. It computes the connected components for graphs where $n \leq \frac{m}{p}$, that is to say for graphs that are relatively dense, and does this without the use of list ranking. Therefore we implemented this part of the algorithm. It computes the connected components of a graph with n vertices and m edges such that $n \leq \frac{m}{p}$ in $\lceil \log p \rceil$ supersteps and $O(\frac{m}{p} + \lceil \log p \rceil n)$ local computations. Each of the p processors requires a memory of $O(\frac{m}{p})$.

We use multi-graphs where two vertices are chosen randomly to form a new edge of the graph. The use of multi-graphs for these tests is not a drawback because the algorithm touches each edge unless it belongs to the spanning tree only once. For this problem, there are two parameters n and m to vary. As the code has the same behavior on the clusters we only show the results for graphs with 1000 and 10 000 vertices on PF. Figure 6 gives the execution times in *seconds per item* with $1, 2, 8$ and 12 PC. For $n = 1000$, m ranges from 10 000 to 500 000. For $n = 10\,000$, m ranges from 10 000 to 36 millions. We see that for a fixed p the curves decrease with m. If we study the results obtained with $n = 1000$ more precisely, we see that when the graph has more than 50 000 edges then there is always a speedup compared to the sequential implementation, and the more processors we use,



(a) $n = 1000$ (b) n=10 000

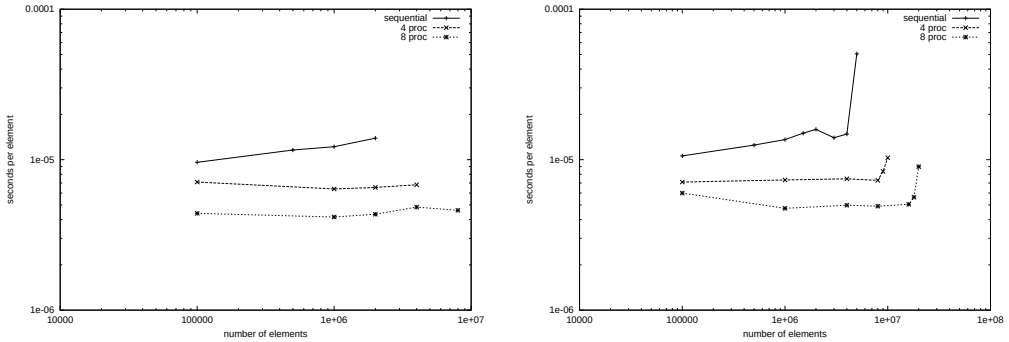
Figure 3: Connected components on PF

the faster is the execution. With $n = 10\,000$, we can do the same remark when the graph has more than 1 million edges. Note, that with $n = 10\,000$ it is possible to handle very large graphs by using several PC. In sequential, the PC begins to swap with about 3.2 millions edges, whereas the connected component computation on a graph with 36 millions edges can be solved in 2.5 seconds with 12 PC.

7 Permutation graphs

The permutation graph associated with a permutation Π is the undirected graph $G = (V,E)$ where $\{i,j\} \in E$ if and only if $i < j$ and $\Pi(i) > \Pi(j)$. Permutations graphs are combinatorial objects that have been intensively studied. Basic references may be found in [Golumbic, 1980]. This graph problem can also be translated into a computational geometry problem called the dominance problem that arises in many applications like range searching, finding maximal elements, interval/rectangle intersection problems ([Preparata and Shamos, 1985]). Passing from the permutation to the graph and vice versa is done easily in a sequential time of $O(n^2)$. In parallel, [Gustedt et al., 1995] show how to pass from the permutation to the graph in the PRAM and their approach easily translates to CGM. This leads to a new compact representation of permutation graphs. The main step of this algorithm is to compute the number of *transpositions* for each value $i = 0,\ldots,n-1$, i.e. the cardinality of $\{j \mid i < j \text{ and } \Pi(i) > \Pi(j)\}$. It requires exactly $\lceil \log_2 p \rceil$ supersteps and $O(\frac{n \log_2 n}{p})$ local computations. The overall communication is in $O(n\lceil \log_2 p \rceil)$ and is then smaller than the local computation cost.

To simplify the implementation and without loss of generality, we assume that p is a power of 2. The generated inputs are random permutations. The elements are unsigned long integers. Figure 4 shows the execution times in *seconds per element* for $1,4$ and 8 PC. For PF, the size of the permutation ranges from 100 000 to 16 millions, whereas for POPC it



(a) POPC (b) PF

Figure 4: A permutation graph algorithm

ranges from 100 000 to 8 millions (due to the memory size of the PC on each cluster). The right end of the curves show the beginning of the swapping effects. First, the curves have the expected behavior: they are constant in n. The execution time is also lowered when we use more processors, as expected. Again, it is possible to solve this problem on very large data. For PF, one PC begins to swap after 4 millions data, whereas 8 PC do it on a little bit less than 17 millions elements. Note that the local computations time is greater than the communications time, as expected.

8 Conclusion

We have presented experimental studies of parallel graphs algorithms in the coarse grained models on clusters. These studies were useful to point out some points concerning the possibilities and the limits of such implementations and the practical use of these coarse grained models on such machines. We can note the following points:

- The curves have the expected behavior.
- It is possible to handle very large data and the network throughput to communicate with other processors is faster than the one for its own disk.
- The two PC clusters do not differ too much and the fact that CGM model does not deal with the conflict problems (that appear in Ethernet network) is justified in our context.
- Memory saturates before the interconnection network and the CGM assumption that considers an unlimited bandwidth is justified for these problems.

We are now going to give some partial answers to the questions we asked at the beginning of the paper.

- Which parallel model can lead to a feasible, portable, predictable and efficient algorithms and code?

Given the analysis of the results, it seems that coarse grained models are very promising

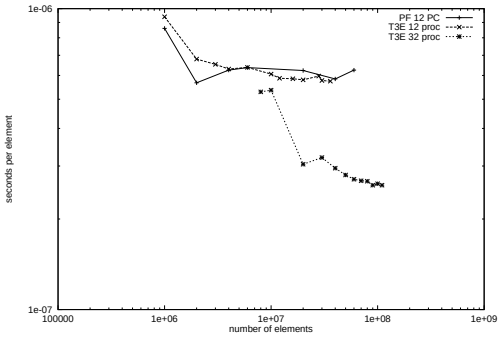


Figure 5: Sort on PF and a T3E

to that regard. If there is still a lot of work left over, these first steps go towards practical and efficient parallel computation.

- What are the possibilities and the limits of the actual graph handling in parallel environments such clusters?

This work shows that it is now possible to write portable code that handles very large data, and that for some problems, this code is efficient. The most challenging problem from the point of view of feasibility and efficiency that we encountered is the list ranking problem. It is possible that this singularity comes from the specific irregular structure of the problem. Nevertheless, it seems obvious that these results can have an impact on many parallel graph algorithms that are based on list ranking.

To conclude and to show that clusters are nowadays powerful systems, Figure 8 compares the execution times for the sort on 12 PC of PF and on 12 and 32 processors of a T3E. This shows that PC clusters are very promising architectures for parallel graph algorithms.

References

[Adler et al., 1995] Adler, M., Byers, J. W., and Karp, R. M. (1995). Parallel sorting with limited bandwith. In *7th ACM Symposium on Parallel Algorithms and Architectures (SPAA'95)*, pages 129–136.

[Anderson and Miller, 1988] Anderson, R. and Miller, G. (1988). Deterministic parallel list ranking. In Reif, J., editor, *Proceedings Third Aegean Workshop on Computing, AWOC 88*, pages 81–90. Springer-Verlag.

[Anderson and Miller, 1990] Anderson, R. and Miller, G. (1990). A simple randomized parallel algorithm for list-ranking. *Information Processing Letter*, 33(5):269–279.

[Baker et al., 1971] Baker, K., Fishburn, P., and Roberts, F. (1971). Partial orders of dimension 2. *Networks*, 2:11–28.

[Bäumer and Dittrich, 1996] Bäumer, A. and Dittrich, W. (1996). Parallel Algorithms for Image Processing: Practical Algorithms with Experiments. In *Proc. of the 10th International Parallel Processing Symposium (IPPS'96)*, pages 429–433.

[Blelloch et al., 1991] Blelloch, G., Maggs, C. L. B., Plaxton, C., Smith, S., and Zagha, M. (1991). A Comparison of Sorting Algorithms for the Connection Machine CM2. In *Symposium on Parallel Algorithms and Architectures (SPAA'91)*, pages 3–16.

[Caceres et al., 1997] Caceres, E., Dehne, F., Ferreira, A., Flocchini, P., Rieping, I., Roncato, A., Santoro, N., and Song, S. W. (1997). Efficient parallel graph algorithms for coarse grained multicomputer and BSP. In *Proceedings of the 24th International Colloquium ICALP'97*, volume 1256 of *LNCS*, pages 390–400.

[Colbourn, 1981] Colbourn, C. (1981). On testing isomorphism of permutation graphs. *Networks*, 11:13–21.

[Cole and Vishkin, 1988] Cole, R. and Vishkin, U. (1988). Approximate parallel scheduling, part I: The basic technique with applications t optimal

parallel list ranking in logarithmic time. *SIAM J. Computing*, 17(1):128–142.

[Cole and Vishkin, 1989] Cole, R. and Vishkin, U. (1989). Faster optimal prefi x sums and list ranking. *Information and Computation*, 81(3):344–352.

[Cormen et al., 1990] Cormen, T., Leiserson, C., and Rivest, R. (1990). *Introduction to Algorithms*. MIT Press.

[Culler et al., 1993] Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K., Santos, E., Subramonian, R., and von Eicken, T. (1993). LogP: Towards a Realistic Model of Parallel Computation. In *Proceeding of 4-th ACM SIGPLAN Symp. on Principles and Practises of Parallel Programming*, pages 1–12.

[Dehne et al., 1993] Dehne, F., Fabri, A., and Rau-Chaplin, A. (1993). Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputer. In *ACM 9th Symposium on Computational Geometry*, pages 298–307.

[Dehne and Götz, 1998] Dehne, F. and Götz, S. (1998). Practical Parallel Algorithms for Minimum Spanning Trees. In *17th IEEE Symposium on Reliable Distributed Systems*, pages 366–371.

[Dehne and Song, 1996] Dehne, F. and Song, S. W. (1996). Randomized parallel list ranking for distributed memory multiprocessors. In Verlag, S., editor, *Proc. 2nd Asian Computing Science Conference ASIAN'96*, volume 1179 of *LNCS*, pages 1–10.

[Dusseau, 1996] Dusseau, A. C. (1996). Fast Parallel Sorting under LogP: Experience with the CM5. *IEEE Transactions on Parallel and Distributed Systems*, 7(8):791–805.

[Ferreira, 1996] Ferreira, A. (1996). Parallel and communication algorithms for hypercube multiprocessors. In Zomaya, A., editor, *Handbook of Parallel and Distributed Computing*. McGraw-Hill.

[Fortune and Wyllie, 1978] Fortune, S. and Wyllie, J. (1978). Parallelism in Random Access Machines. In *10-th ACM Symposium on Theory of Computing*, pages 114–118.

[Gerbessiotis and Siniolakis, 1996] Gerbessiotis, A. V. and Siniolakis, C. J. (1996). Deterministic Sorting and Randomized Median Finding on the BSP model. In *8th ACM Symposium on Parallel Algorithms and Architectures (SPAA'96)*, pages 223–232.

[Gerbessiotis and Valiant, 1994] Gerbessiotis, A. V. and Valiant, L. G. (1994). Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing*, 22:251–267.

[Golumbic, 1980] Golumbic, M. C. (1980). *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

[Goodrich, 1996] Goodrich, M. (1996). Communication-effi cient parallel sorting. In *Proc. of 28th Symp. on Theory of Computing*.

[Greiner, 1994] Greiner, J. (1994). A Comparison of Parallel Algorithms for Connected Components. In *6th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'94)*, pages 16–25.

[Guérin Lassous, 1999] Guérin Lassous, I. (1999). *Algorithmes parallèles de traitement de graphes : une approche basée sur l'analyse expérimentale*. PhD thesis, Université Paris 7.

[Guérin Lassous and Gustedt, 2000] Guérin Lassous, I. and Gustedt, J. (2000). List ranking on PC clusters. Technical Report 3869, I.N.R.I.A.

[Guérin Lassous and Morvan, 1998] Guérin Lassous, I. and Morvan, M. (1998). Some results on ongoing research on parallel implementation of graph algorithms: the case of permutation graphs. In H.R Arabnia, e., editor, *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 325–330.

[Guérin Lassous and Thierry, 2000] Guérin Lassous, I. and Thierry, É. (2000). Generating random permutations in the parallel coarse grained models framework. Working paper.

[Gustedt et al., 1995] Gustedt, J., Morvan, M., and Viennot, L. (1995). A compact data structure

and parallel algorithms for permutation graphs. In Nagl, M., editor, *WG '95 21st Workshop on Graph-Theoretic Concepts in computer Science*. Lecture Notes in Computer Science 1017.

[Hsu et al., 1995] Hsu, T.-S., Ramachandran, V., and Dean, N. (1995). Implementation of parallel graph algorithms on a massively parallel SIMD computer with virtual processing. In *Proc. 9th International Parallel Processing Symposium*, pages 106–112.

[Hsu et al., 1997] Hsu, T.-S., Ramachandran, V., and Dean, N. (1997). Parallel Implementation of Algorithms for Finding Connected Components in Graphs. In *Parallel Algorithms : Third DIMACS Implementation Challenge*, volume 30 of *DIMACS Series*, pages 395–416. American Math. Soc.

[Jájá, 1992] Jájá, J. (1992). *An Introduction to Parallel Algorithm*. Addison Wesley.

[Karp and Ramachandran, 1990] Karp, R. and Ramachandran, V. (1990). Parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science Volume A: Algorithms and Complexity*, pages 869–942. Elsevier.

[Krishnamurthy et al., 1997] Krishnamurthy, A., Lumetta, S., Culler, D. E., and Yelick, K. (1997). Connected Components on Distributed Memory Machines. In Bhatt, S., editor, *Third DIMACS Implementation Challenge*, volume 30 of *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, pages 1–21.

[Krizanc and Saarimaki, 1999] Krizanc, D. and Saarimaki, A. (1999). Bulk synchronous parallel: practical experience with a model for parallel computing. *Parallel Computing*, 25:159–181.

[Kumar et al., 1997] Kumar, S., Goddard, S. M., and Prins, J. F. (1997). Connected-Components Algorithms for Mesh-Connected Parallel Computers. In Bhatt, S., editor, *Third DIMACS Parallel Challenge*. Academic Press.

[Leighton, 1992] Leighton, F. T. (1992). *Introduction to Parallel Algorithms and Architectures: Arrays . Trees . Hypercubes*. Morgan Kaufmann.

[Lumetta et al., 1995] Lumetta, S. S., Krishnamurthy, A., and Culler, D. E. (1995). Towards Modeling the Performance of a Fast Connected Components Algorithm on Parallel Machines. In *Proceedings of Supercomputing'95*, San Diego, California.

[Pneuili et al., 1971] Pneuili, A., Lempel, A., and Even, W. (1971). Transitive orientation of graphs and identifi cation of permutation graphs. *Canad. J. math*, 23:160–175.

[Preparata and Shamos, 1985] Preparata, F. and Shamos, M. (1985). *Computational Geometry: an Introduction*. Springer-Verlag.

[Reid-Miller, 1994] Reid-Miller, M. (1994). List ranking and list scan on the Cray C-90. In *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pages 104–113.

[Sibeyn, 1997] Sibeyn, J. F. (1997). Better Trade-offs for Parallel List Ranking. In *Proceedings of the 9th ACM Symposium on Parallel Algorithms and Architectures (SPAA'97)*, pages 221–230.

[Sibeyn et al., 1999] Sibeyn, J. F., Guillaume, F., and Seidel, T. (1999). Practical Parallel List Ranking. *Journal of Parallel and Distributed Computing*, 56:156–180.

[Spinrad and Valdes, 1983] Spinrad, J. and Valdes, J. (1983). Recognition and isomorphism of two dimensional partial orders. In 10th Coll. on Automata, Language and Programming., number 154 in Lecture Notes in Computer Science, pages 676–686. Springer-Verlag.

[Tanenbaum, 1997] Tanenbaum, A. (1997). *RESEAUX*. Prentice Hall, 3ème edition.

[Valiant, 1990] Valiant, L. (1990). A bridging model for parallel computation. *Communications of the ACM*, Vol. 33(8):103–111.

[Wyllie, 1979] Wyllie, J. (1979). *The Complexity of Parallel Computations*. PhD thesis, Computer Science Department, Cornell University, Ithaca, NY.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399